ASSIGNMENT-01

SUBMISSION BY

JANNATUL FERDAUS OISHI

2022-3-60-216

COURSE:CSE246

SECTION:01

SUBMITTED TO

Dr. Taskeed Jabid

Professor

Department of Computer Science & Engineering

SUBMISSION DATE

30 NOVEMBER,2024

QUESTION:01

```
#include <stdio.h>

void divider(int arr[], int aux[], int low, int high)
{
   if (high == low)
   {
      return;
   }

   int mid = (low + high) / 2;
   divider(arr, aux, low, mid);      // split/merge the left half
   divider(arr, aux, mid + 1, high);  // split/merge the right half
   merge(arr, aux, low, mid, high);   // merge the two half runs
}

void merge(int arr[],int aux[],int low,int mid,int high)
{
   int k=low,i=low,j=mid+1;
   while(i<=mid && j<=high)
   {
      if(arr[i]<=arr[j])
      {
         aux[k++]=arr[i++];
      }
      else
      {
         aux[k++]=arr[j++];
      }
   }

   while(i<=mid)
   {
```

```
      aux[k++]=arr[i++];
   }

    while (j <= high) {
      aux[k++] = arr[j++];
   }

   for(int i=low;i<=high;i++)
   {
      arr[i]=aux[i];
   }

}


// Function declarations here (for divider and merge)

int main() {
   int n;
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int a[n], aux[n];  // Declare both 'a' and 'aux' arrays

   for (int i = 0; i < n; i++) {
      a[i]=rand()%1000; //Test sorting algorithms on unpredictable data and Avoiding
Hardcoded Data
   }

   divider(a, aux, 0, n - 1);

   printf("\n\nAfter implementing Merge sort, Sorted List is :: \n\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", a[i]);
   }
   return 0;
}
```

QUESTION-2

```
#include <stdio.h>

void swap(int a[], int i, int j) {
   int temp = a[i];
   a[i] = a[j];
   a[j] = temp;
}

int partition(int A[], int p, int r) {
   int x = A[r];  // pivot element
   int i = p - 1;
   for (int j = p; j <= r - 1; j++) {
      if (A[j] <= x) {
         i++;
         swap(A, i, j);
      }
   }
   swap(A, i + 1, r);
   return i + 1;
}

void quicksort(int A[], int p, int r) {
   if (p < r) {
      int q = partition(A, p, r);
      quicksort(A, p, q - 1);
      quicksort(A, q + 1, r);
   }
}

void printArray(int A[], int size) {
   for (int i = 0; i < size; i++) {
```

```
    printf("%d ", A[i]);
  }
  printf("\n");
}

int main() {
  int A[] = {12, 7, 14, 9, 10, 11};
  int size = sizeof(A) / sizeof(A[0]);

  printf("Original array: ");
  printArray(A, size);

  quicksort(A, 0, size - 1);

  printf("Sorted array: ");
  printArray(A, size);

  return 0;
}
```

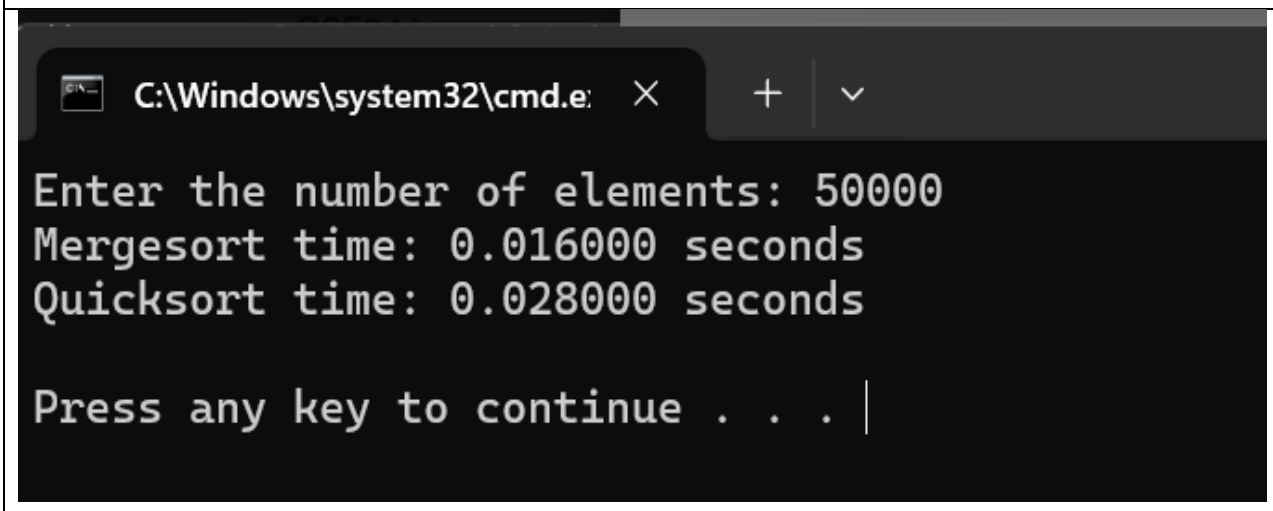
TASK-01

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


void mergesort(int arr[], int aux[], int low, int high)
{
  if (high == low)
  {
    return;
  }

  int mid = (low + high) / 2;
  mergesort(arr, aux, low, mid);      // split/merge the left half
  mergesort(arr, aux, mid + 1, high);  // split/merge the right half
  merge(arr, aux, low, mid, high);   // merge the two half runs
```

```
}

void merge(int arr[], int aux[], int low, int mid, int high)
{
   int k = low, i = low, j = mid + 1;
   while (i <= mid && j <= high)
   {
      if (arr[i] <= arr[j])
      {
         aux[k++] = arr[i++];
      }
      else
      {
         aux[k++] = arr[j++];
      }
   }

   while (i <= mid)
   {
      aux[k++] = arr[i++];
   }

   while (j <= high)
   {
      aux[k++] = arr[j++];
   }

   for (int i = low; i <= high; i++)
   {
      arr[i] = aux[i];
   }
}

void swap(int arr[], int i, int j) {
   int temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
}

int partition(int arr[], int low, int pivot) {
   int x = arr[pivot];  // Choose the pivot element
   int i = low - 1;     // Start with i one position before low

   for (int j = low; j <= pivot - 1; j++) {
```

```
      if (arr[j] <= x) {
        i++;
        swap(arr, i, j);  // Place smaller elements to the left
      }
    }

    swap(arr, i + 1, pivot);  // Place pivot in the correct position
    return i + 1;  // Return the position of the pivot
}

void quicksort(int arr[], int low, int r) {
    if (low < r) {
        int pivot = partition(arr, low, r);  // Partition the array
        quicksort(arr, low, pivot - 1);     // Sort left part
        quicksort(arr, pivot + 1, r);       // Sort right part
    }
}

int main()
{
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr1[n], arr2[n], aux[n]; // arr1 and aux for merge sort, arr2 for quicksort
    for (int i = 0; i < n; i++)
    {
        arr1[i] = rand() % 1000;
    }

    for (int i = 0; i < n; i++)
    {
        arr2[i] = arr1[i]; // Copy the array for both sorts
    }

    clock_t start, end;

    // Mergesort time complexity
    start = clock();
    mergesort(arr1, aux, 0, n - 1); // Call divider instead of mergesort
    end = clock();
    double mergeSortTime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Mergesort time: %lf seconds\n", mergeSortTime);
```

```
    // Quicksort time complexity
    start = clock();
    quicksort(arr2, 0, n - 1); // Sort arr2 with quicksort
    end = clock();
    double quickSortTime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Quicksort time: %lf seconds\n", quickSortTime);

    return 0;
}
```



TASK-02

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void merge(int arr[], int aux[], int low, int mid, int high)
{
    int k = low, i = low, j = mid + 1;
    while (i <= mid && j <= high)
    {
        if (arr[i] >= arr[j])
        {
            aux[k++] = arr[i++];
        }
        else
        {
            aux[k++] = arr[j++];
```

```
    }
  }

  while (i <= mid)
  {
    aux[k++] = arr[i++];
  }

  while (j <= high)
  {
    aux[k++] = arr[j++];
  }

  for (int i = low; i <= high; i++)
  {
    arr[i] = aux[i];
  }
}

void divider(int arr[], int aux[], int low, int high)
{
  if (high == low)
  {
    return;
  }

  int mid = (low + high) / 2;
  divider(arr, aux, low, mid);      // split/merge the left half
  divider(arr, aux, mid + 1, high);  // split/merge the right half
  merge(arr, aux, low, mid, high);   // merge the two half runs
}



void swap(int arr[], int i, int j) {
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}

int partition(int arr[], int low, int pivot) {
  int x = arr[pivot];  // Choose the pivot element
  int i = low - 1;     // Start with i one position before low
```

```
    for (int j = low; j <= pivot - 1; j++) {
        if (arr[j] <= x) {
            i++;
            swap(arr, i, j);  // Place smaller elements to the left
        }
    }

    swap(arr, i + 1, pivot);  // Place pivot in the correct position
    return i + 1;  // Return the position of the pivot
}

void quicksort(int arr[], int low, int r) {
    if (low < r) {
        int pivot = partition(arr, low, r);  // Partition the array
        quicksort(arr, low, pivot - 1);     // Sort left part
        quicksort(arr, pivot + 1, r);       // Sort right part
    }
}



int main()
{
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr1[n], arr2[n], aux[n];
    for (int i = 0; i < n; i++)
    {
        arr1[i] = rand() % 1000;
    }

    for (int i = 0; i < n; i++)
    {
        arr2[i] = arr1[i];
    }

    clock_t start, end;

    // Mergesort time complexity
    start = clock();
    divider(arr1, aux, 0, n - 1);
    end = clock();
```

```
    double mergeSortTime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Mergesort time: %lf seconds\n", mergeSortTime);

    // Quicksort time complexity
    start = clock();
    quicksort(arr2, 0, n - 1);
    end = clock();
    double quickSortTime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Quicksort time: %lf seconds\n", quickSortTime);

    return 0;
}
```

```
Enter the number of elements: 30000
Mergesort time: 0.004000 seconds
Quicksort time: 0.020000 seconds

Process returned 0 (0x0)   execution time : 31.100 s
Press any key to continue.
```